OIPE JCS5
AUG 0 3 2004
PATENT & TRADEMARK OFFICE

Cisco Systems

Document Number	ENG-36187
Revision	1
Author	Tom Edsall, Silvano Gai
Project Manager	Tom Edsall

RegEx: Regular Expression Logic

This document describes the RegEx (Regular Expression Logic) developed for Asti. Asti is a board designed to support the NSSA (Network Services Software Architecture) and IPv6 routing. The RegEx logic implements packet parsing using regular expression matching, field extraction and subroutine call.

Table 1: Project Headline Approvals

Name	Approval Date	Name	Approval Date

Modification History

Rev	Date	Originator	Comment
1		S.Gai	Initial Draft

Table of Contents

1.0 Introduction

Asti is a card for the Catalyst 6000 designed to support NSSA (Network Services Software Architecture) and IPv6 routing [1]. In particular Asti addresses the following issues:

- IPv6 routing. Unicast (128 bits), Multicast (256 bits) and ACL (approximately 380 bits) lookups;
- SLB (Server Load Balancing), both SM (Service Manager) and FA (Forwarding Agent);
- IP reassembly;
- TCP termination and resequencing;
- Intrusion Detection;
- Web Cache Redirection, using WCCP or CASA-R;
- Firewall;
- NAT;
- RMON/NetFlow/Performance Monitoring;
- Traffic Shaping [2].

This document describes the RegEx (Regular Expression) Logic developed for Asti. The RegEx Logic is a highly programmable logic circuit, based on a TCAM, designed to speed up the parsing of packets up to the application layer. The performance target for RegEx is to parse flows at multi-Gigabit/s.

RegEx is the logical continuation of the research activity in the TCAM field. Up to now, TCAMs have been deployed with success as combinatorial circuits, i.e. a packet is given in input to a TCAM and this can match or not one or more TCAM rows.

RegEx can be seen either as a complex state machine built using a TCAM as combinatorial logic or as a packet processor designed to perform efficient pattern matching.

The concept of a programmable state machine is very powerful: eight TCAM2 can be cascaded to obtain 128K entries, each 288 bit wide. Each bit may assume a ternary value (0, 1, don't care). Therefore the realization of powerful state machine for pattern matching becomes very easy. Moreover TCAM2 can perform 100 million lookups per second and therefore also the speed of the state machine is very high. The speed can be improved having a single large TCAM shared by 4 sequential logic circuits, each operating on a different packet. This to compensate the pipeline architecture of the TCAM, that requires multiple clock cycles to compute a result.

The concept of packet processor for pattern matching is equally powerful. The state of the state machine can be seen as a program counter. It is possible to augment the state machine with a stack to introduce subroutine calls. Operators to redirect packets to an external CPU, to rewrite them, and send them out, or to continue the parsing, are possible. Fields can be extracted from the packets and used in subsequent matches and computations. Finally matches can be implemented easily, since they are intrinsic in the ternary architecture of the TCAM.

The RegEx logic uses a different number of clock cycles to process different types of packets. This is considered acceptable for NSSA services, and the Asti board is designed to operate in store-and-forward mode, i.e. the packets cross the buss twice.

2.0 Requirements for RegEx

The main requirement for RegEx is the development of a highly programmable packet parsing logic, not dedicated to the solution of a single problem. To achieve this goal we had a look to well known packet parsing problem such as:

- Ethenet v.2 encapsulation vs. LLC/SNAP vs. Novel LLC;
- Variable header length processing;
- TLV parsing;

- option parsing (e.g., TCP, IP options);

and also to more recent parsing problem, such as:

- MIME parsing (RFC 1521) for email, http, etc.;
- {keyword, value} parsing with value extraction;
- text file parsing with either:
    - exact match;
    - regular expression match.

The parsing can be done on packets, packet fragments or TCP stream. RegEx does not perform IP packet reassembly or TCP resequencing, if these functions are required, they must be performed prior to RegEx [1], [3]. There is not an intrinsic limit in the depth of the parsing.

While RegEx parses a packet it can extract the value associated with some fields to simplify the decision process of a CPU, redirect the packet to the CPU pre-pending the extracted fields, redirect the packet to the network associating a rewrite rule, increment counters and call subroutines.

The RegEx logic is able to take the final decision for most of the packets, but not for all of them. In fact, some packets may require a decision based on the execution of a software algorithm that is going to change over time (a classical example is the server load balancing algorithm that can take its decision looking to fields such as IP addresses, SSL-ID, cookies, etc.). In this case the RegEx logic extracts the fields important for the decision, pre-pend them to the packet and send the packet to a general purpose CPU for the final decision.

3.0 RegEx and Asti

Figure 1 shows the block diagram of the Asti board. The Regex ASIC is outlined in gray together with the TCAM and the SRAM that it needs. As the block diagram shows, the design has well defined interfaces and this simplifies the reuse of this ASIC in other platforms like the 8500.

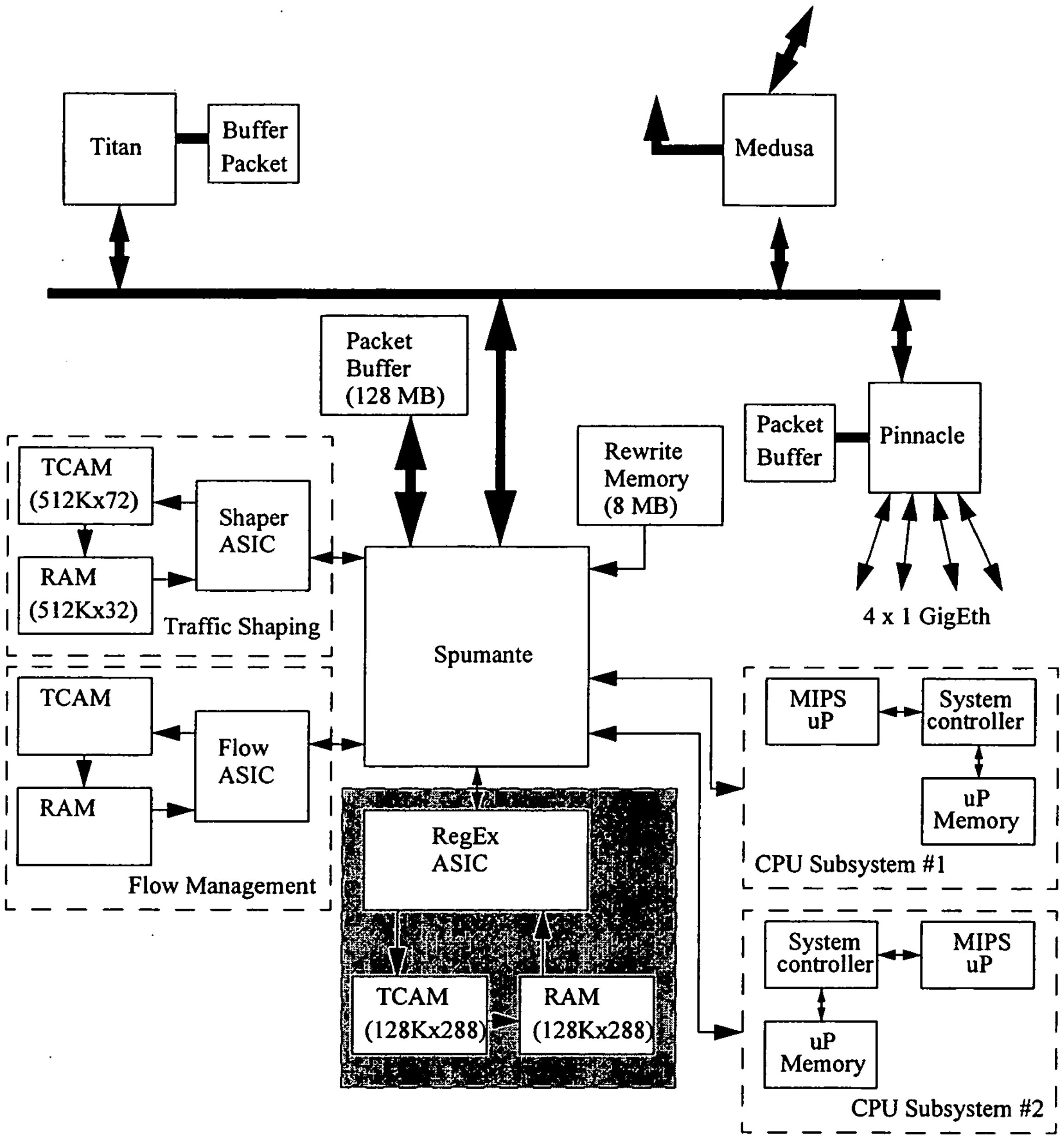


Figure 1. Asti Block Diagram

4.0 The architecture of RegEx

Figure 2 details the connections between the Asti main ASIC (Spumante) and the RegEx ASIC. The main connections are for transferring the packet from the Asti packet buffer to the RegEx packet buffer and to transfer back the parsed information. Few control signals are also required for synchronization and to pass back simple decisions.

The RegEx Logic uses an external TCAM to store the parsing rules and the TCAM has an associated SSRAM.

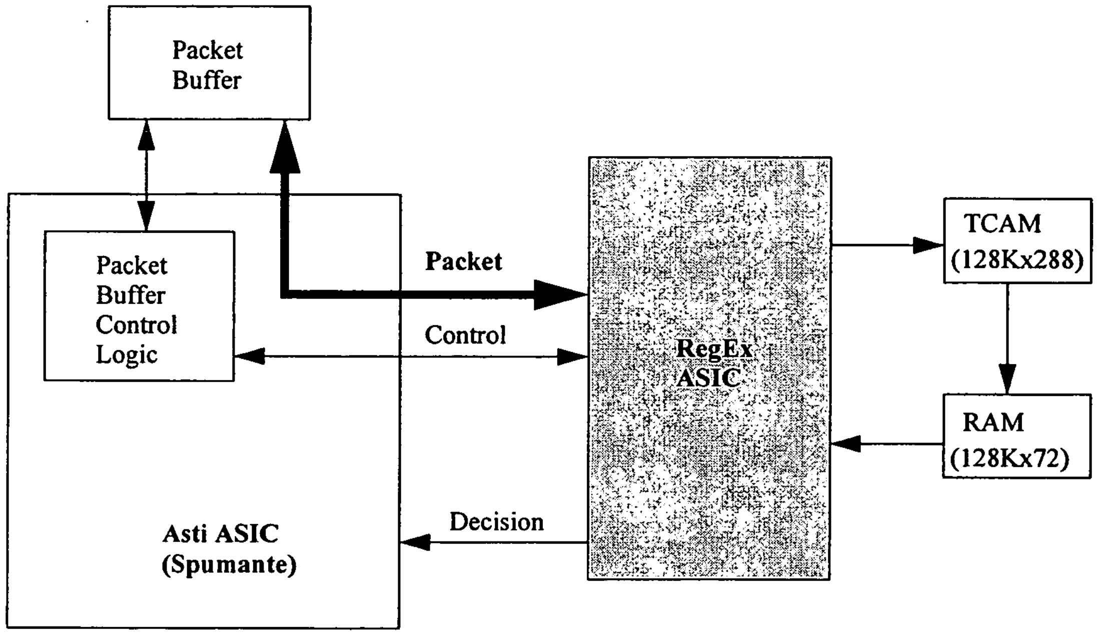


Figure 2. Connections between the Asti ASIC and the RegEx ASIC

Figure 3 details the internal architecture of the RegEx ASIC. The main functional blocks are:

- the packet pre-parser;
- the single packet buffer;
- the barrel shifter;
- the input register (including the tag register);
- the decoder;
- the counter RAM;
- the stack;
- the field RAM;
- the interfaces to the external TCAM and SSRAM.

To improve the speed of the RegEx ASIC, most of the logic is present in four instances, so that four packets can be in processing at the same time. This is done to compensate the pipeline architecture of the TCAM, that requires more than one clock cycle to compute a result.

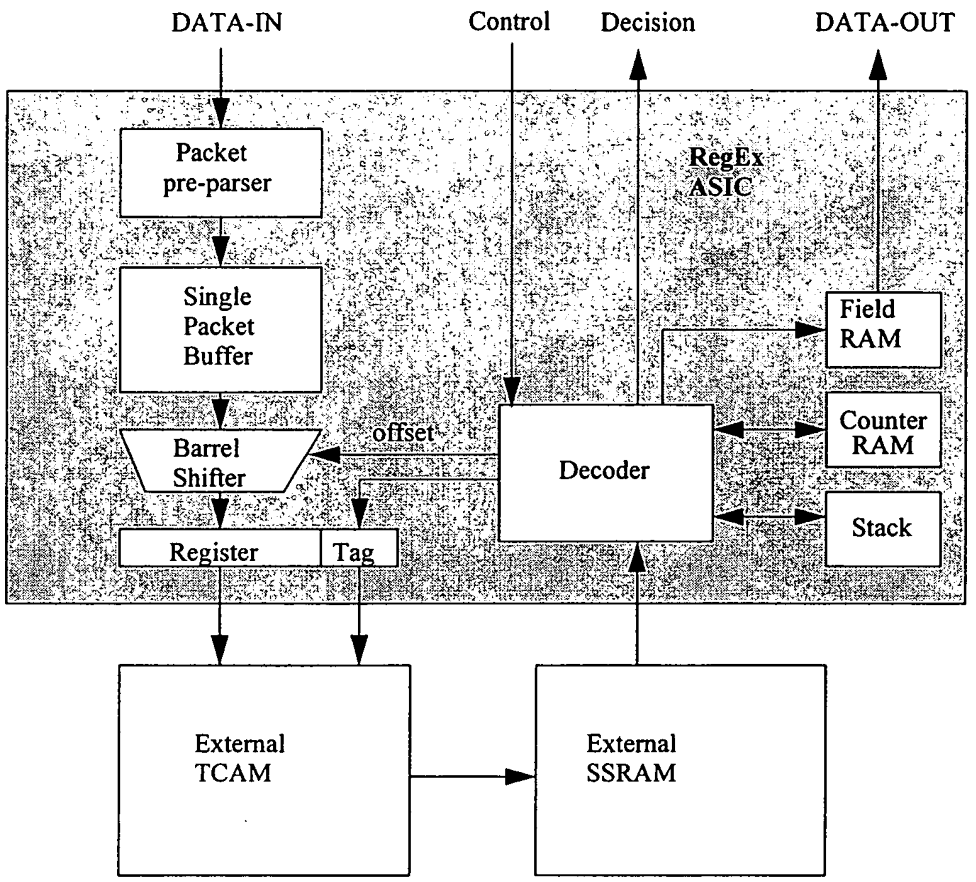


Figure 3: The RegEx ASIC

4.1 The Packet pre-parser

To simplify the job of the RegEx logic and to speed-up the processing of the most common cases, the packet, before being stored in the single packet buffer, is preprocessed with conventional techniques to extract the most commonly used fields.

The extracted fields (see Figure 4) are pre-pended to the packet in the single packet buffer.

```
/*
    *  Prepended Cisco specific stuff
    *                                  length      byte-index
    *                               bits    bytes
    *  IP Source Address             32       4        0
    *  IP Destination Address        32       4        4
    *  TCP/UDP Source Port           16       2        8
    *  TCP/UDP Destination Port      16       2       10
    *  IP Protocol Type               8       1       12
    *  IP TOS                         8       1       13
    *  VLAN-ID                       12       2       14
    *  TCP ACK Flag                   1                msb-3 of byte 15
    *  TCP SYN Flag                   1                msb-2 of byte 15
    *  TCP FIN Flag                   1                msb-1 of byte 15
    *  TCP RST Flag                   1                msb   of byte 15
    *  Input Port Index              19       3       16
    * 0 pad                           5                msb-4 of byte 18
    * Layer 2 packet                   VARIABLE       19
    */
```

Figure 4. Pre-pendent Information

4.2 The Single Packet Buffer

The Single Packet buffer is a memory where either a packet, or a packet fragment, or a TCP byte stream is stored while RegEx is processing it. A packet received by the bus is stored in the single packet buffer after pre-pending the information of Figure 4.

The information stored in the Single Packet Buffer undergoes one or more look-up into the TCAM under the control of the decoder. In general 32 bytes a time are looked-up in the TCAM

4.3 The Barrel Shifter

The Barrel Shifter is capable of shifting the packet in increments of one byte and therefore is able to select which 32 bytes in the single packet buffer should be matched with the TCAM. The Barrel Shifter is controlled by the decoder that provide the offset into the packet. The output is concatenated with the tag value and some other fields and presented to the external TCAM.

If there are 32 bytes (256 bits) of data in the single packet buffer, the format of Figure 5(a) is used and the field valid contains the value zero. If there are N bytes (with N less than 32), the format of Figure 5(b) is used and the field valid contains the number N.

The E (End) bit is used to indicate that the Barrel Shifter has performed the last possible shift (end of packet reached).

The Tag value is the state of RegEx. It is used to concatenate multiple lookups into a single lookup.

In general, the successive lookups performed on the same packet differ for the offset and for the tag.

1	**5**	**6**	**20**	**256**	**bits**
E	**Valid**	**Not Used**	**Tag**	**Packet Data**	

(a)

1	**5**	**6**	**20**	**N**	**256 - N**	**bits**
E	**Valid**	**Not Used**	**Tag**	**Packet Data**	**Padding**	

(b)

Figure 5. TCAM Input

For each lookup the TCAM return the index of the first matching entry (it is possible to assume that the last row of a TCAM is a match all entry) and this index is used to address the SSRAM.

4.4 The TCAM and SSRAM

The TCAM subsystem uses eight TCAM2s (ENG-26968). It is possible to reach 128K entries, each 288 bits wide connecting eight TCAM2. The TCAMs are associated with a RAM (SSRAM) 128K x 72 bits.

The TCAMs entries can be inserted/deleted by the Catalyst 6000 main processor or by the CPU sub-systems.

The SSRAM returns 72 bits with the general format shown in Figure 6.

1	6	65	bits
D	Opcode	Instruction Specific Data	

Figure 6: From SSRAM Data Format

The meaning of the different fields is the following:

- D (Done). Done bit, it indicates that the packet requires no further processing from the regular expression logic.
- Opcode. It indicates the action to be taken. The following values are possible:

```
Opcode encoding:
```

- Instruction Specific Data: depends from the Opcode, as specified in Figure 7.

CONT (Continue TCAM lookup)

1	6	1	8	16	20	20	bits
D	CONT	R	Offset	Not Used	Not Used	Tag	

NTWK (Redirect to Network)

1	6	1	8	16	20	20	bits
D	NTWK	R	Offset	SID	RW Pointer	Tag	

CPU (Redirect a CPU)

1	6	1	8	16	20	20
D	CPU	R	Offset	SID	Progr. Pointer	Tag

CNT (Counter)

1	6	1	8	16	20	20	bits
D	CNT	R	Offset	Value	Counter No.	Tag	

CALL (Subroutine Call)

1	6	1	8	16	20	20	bits
D	CALL	R	Offset	Not Used	Return Tag	Tag	

RET (Subroutine Return)

1	6	1	8	16	20	20	bits
D	RET	R	Offset	Not Used	Not Used	Not Used	

MARK (the current Offset)

1	6	1	8	8	8	20	20	bits
D	MARK	R	Offset	NU	OA	TBD	Tag	

MEMCPY (Memory Copy)

1	6	1	8	8	8	12	8	20	bits
D	MEMCPY	R	Offset	FN	OA	NU	DI	TBD	

OFS (Shift with Variable Offset)

1	6	1	8	12	1	3	8	4	8	20	bits
D	OFS	R1	OP	NU	R2	OL	NU	OG	OC	Tag	

NU: Not Used

Figure 7. Opcode specific information

4.5 The Opcodes

This section contains a brief description of the opcodes. The fields with the same name have the same meaning in different opcodes and are described only in the first one.

The opcodes currently defined are:

```
CONT = 0   /*    continue matching with the next tag*/
NTWK = 1   /*    send this packet to the network */
CPU  = 2   /*    send this packet to the CPU */
CNT  = 3   /*    increment a counter */
CALL = 4   /*    call a subroutine */
RET  = 5   /*    return from a subroutine */
MARK = 6   /*  save the current offset */
MEMCPY = 7 /*  copy information about a field into the field RAM */
OFS  = 8   /*    shift with a variable offset */
```

4.5.1 CONT (Continue)

Perform another lookup on the packet. This opcode is useful when a long pattern match must be split into shorter ones (e.g. IPv6 ACLs). The **Offset** field contains the offset for the Barrel Shifter. The Offset can be encoded as absolute or relative, depending on the **R** bit (R = 1, the offset is relative. If relative the offset is encoded as a signed integer in two-complement format.

The **Tag** field is the tag to be used in the next lookup (future state of the state machine). The Tag itself has the format shown in Figure 8.

TBD: with 8 bits of offest the absolute offset is limite to 255. Is this acceptable?

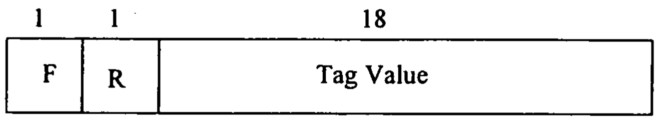


Figure 8. Tag Format

- F: Flag. It is the MSB and it will be always set to 0. F=1 is reserved for future use.
- R: Indicates if the TAG is to be considered Absolute (R=0) or Relative (R=1). If R=1, the Tag Value is added to the current Tag (the one used for the last lookup) and the result is used as a Tag for the next lookup.
- Tag Value: Contains the tag value. If R=1 Tag Value is a 2-complement number.

4.5.2 NTWK (Network)

Send this packet to the network. The **RW (ReWrite) pointer** points to a rewrite rule in the Rewrite Memory (see Figure 1). The packet will be rewritten according to the content of the rewrite memory pointed by RW.

The packet is also shaped (see ENG-35513) using the shaper indexed by **SID (Shaper ID).**

4.5.3 CPU

Send this packet to one of the CPUs. The **Prog. (Program) Pointer** points to a program in the CPU memory. The packet will be processed by a CPU subsystem using the program pointed by

Pointer. For example, if the TCAM has matched an H.225 packet (the control session of H.323), the packet is passed to the CPU subsystem together with the pointer to the routine to perform stateful inspection of H.323. The routine executes in the CPU subsystem, it learns the parameters on the H.323 session, and may insert additional entries in the TCAM to match the H.323 packets.

4.5.4 CNT (Count)

Increment a counter. The value **Value** is summed to the Counter indexed by **Counter No.** (Counter Number). Value is in two-complement format.

4.5.5 CALL

This instruction is similar to the CONT instruction, but it also pushes the **Return Tag** contained in the instruction on the stack (see Figure 3). This is useful to implement a call to subroutine mechanism.

4.5.6 RET (Return)

This instruction is similar to the CONT instruction, but it pops the TAG from the stack.

4.5.7 MARK

This opcode is used to save in a register of RegEx the current offset. Before saving it, the value **OA (Offset Adjustment)** is added to the current offsent. OA is a signed integer in two-complement format.

4.5.8 MEMCPY (Memory Copy)

The Field RAM contains an array of field descriptor (see Section 4.8). This opcode is used to copy in the Field RAM a field descriptor. Each field descriptor is composed of three items:

- a name on 8 bits;
- a length on 8 bits;
- an absolute offset on 16 bits.

The value **OA (Offset Adjustment)** is added to the current offsent and used as the end field pointer. The begin field pointer must be saved with a MARK instruction. The length is computed automatically. The name can be specified using the **FN (Field Name)** operand. These values are saved in the descriptor with index **DI (Descriptor Index).**

4.5.9 OFS (Offset)

The Offset Instruction computes an offset to be used in the next lookup, starting from a field in the packet. The **OP (Offset Position)** points to the most significative byte of the offset, **OL (Offset Length)** contains the length of the offset in byte (legal values are 7 most significative nibble), 0 (least significative nibble) 1 one byte, 2 two bytes, 4 four bytes) **OG (Offset Granularity)** contains the granularity of the offset (0 means 1 nibble, 1 means 1 byte unit, 2 means 16 bit unit, 4 means 32 bit units, etc.). Moreover an **OC (Offset Constant)** can be added.

R1 means Relative/Absolute Flag for Offset_Pos.

R2 means Relative/Absolute Flag for Offset.

4.6 The decoder Logic

The decoder logic receives the result from the SSRAM.

If the Done bit is set, the packet is either discarded or redirected to the CPU or to the network, possibly pre-pending the information in the field RAM.

If the Done bit is not set, the decoder shifts the input to the TCAM according to the R and Offset fields and set the new TAG value.

4.7 The Counter RAM

The counter RAM is a RAM that contains a set of counters. The instruction CNT can increment these counters. TBD: how do we initialize these counters? how do we read them?

4.8 The Field RAM

The field RAM is a RAM that contains field descriptor created using the MEMCPY instruction. The format is reported in Figure 9.

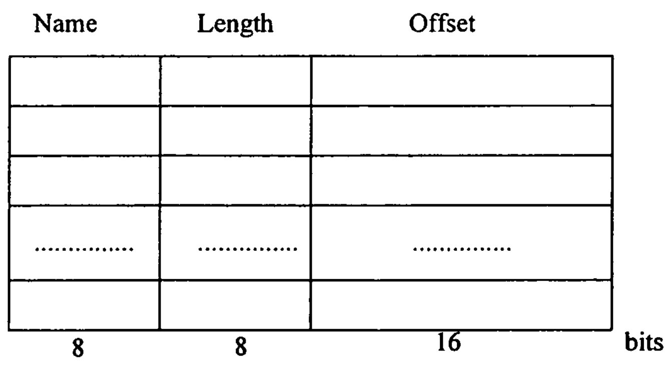


Figure 9. The Field RAM

4.9 The Stack

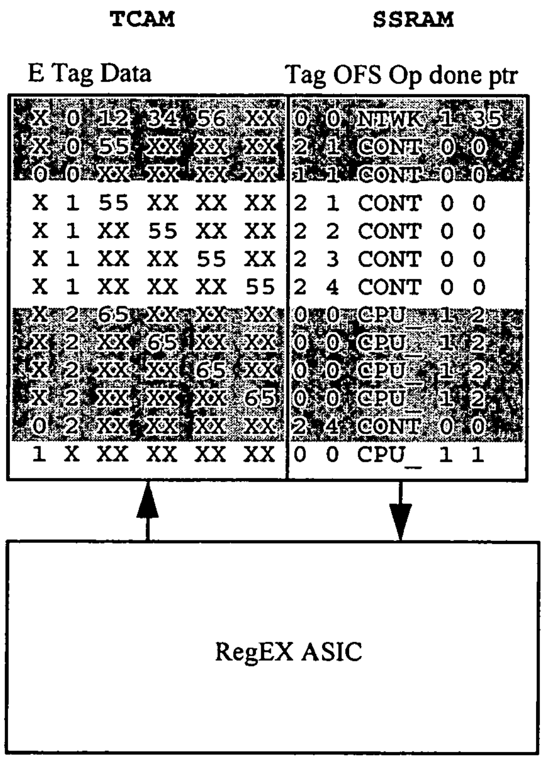
The stack is a 64 entries stack pointed by a stack pointer. It is used to stack the tag values in assoxciation with the instructions call and return.

5.0 Regular expression example

Figure 11 shows the contents of the TCAM and the SSRAM to perform the match of Figure 10.

```
if (packet contains 123456*) then
     send to network with rewrite = 35
else if (packet contains *55*65) then
     send to CPU program #2
else send to CPU program #1
```

Figure 10. Example of Regular Expression



TCAM						SSRAM				
E	Tag	Data				Tag	OFS	Op	done	ptr
X	0	12	34	56	XX	0	0	NTWK	1	35
X	0	55	XX	XX	XX	2	1	CONT	0	0
0	0	XX	XX	XX	XX	1	1	CONT	0	0
X	1	55	XX	XX	XX	2	1	CONT	0	0
X	1	XX	55	XX	XX	2	2	CONT	0	0
X	1	XX	XX	55	XX	2	3	CONT	0	0
X	1	XX	XX	XX	55	2	4	CONT	0	0
X	2	65	XX	XX	XX	0	0	CPU_	1	2
X	2	XX	65	XX	XX	0	0	CPU_	1	2
X	2	XX	XX	65	XX	0	0	CPU_	1	2
X	2	XX	XX	XX	65	0	0	CPU_	1	2
0	2	XX	XX	XX	XX	2	4	CONT	0	0
1	X	XX	XX	XX	XX	0	0	CPU_	1	1

Figure 11: Implementation of the previous example

6.0 Expected performance

The regular expression logic has the following expected performance:

- IPv4 flow match: 50 Mpps
- IPv6 flow match: 16.7 Mpps
- Arbitrary pattern 64 byte frames - 16.7 Mpps
    - Looking at 32 bytes at a time
    - Equivalent to 8.5 Gigabit/s
- Arbitrary Pattern 1500 byte frames - 1 Mpps

- Looking at 32 bytes at a time
- Equivalent to 12 Gigabit/s

7.0 Bibliography

[1] Tom Edsall, Silvano Gai, "Asti," ENG-35512.

[2] Silvano Gai, "Shaping in Asti," ENG-35513.

[3] Dante Malagrino', "Hardware IP reassembly in Asti," ENG-35737.